

# POCKET COMPUTER

## NEWSLETTER



© Copyright 1983 — Issue 22

February

### TEXAS INSTRUMENTS JOINS FRAY WITH THE CC-40

Calling its entry to the "portable" computer market the Compact Computer 40 (abbreviated CC-40), Texas Instruments joined the throng of traditional calculator manufacturers who are providing true computer power in miniaturized form.

The TI CC-40 weighs in at 22 ounces with dimensions of 9-1/2 by 5-3/4 by 1 inch (remarkably close to Hewlett-Packard's HP-75). It can be powered for approximately 200 hours on a set of four AA alkaline batteries. An optional 115-volt ac adapter provides desktop operation if desired.

The little computer has a 31-character, scrollable liquid crystal display that presents both upper and lower case letters. There are 18 built-in annunciators including 6 user-settable flags. The staggered keyboard layout in QWERTY format and a numeric keypad provide for ease in

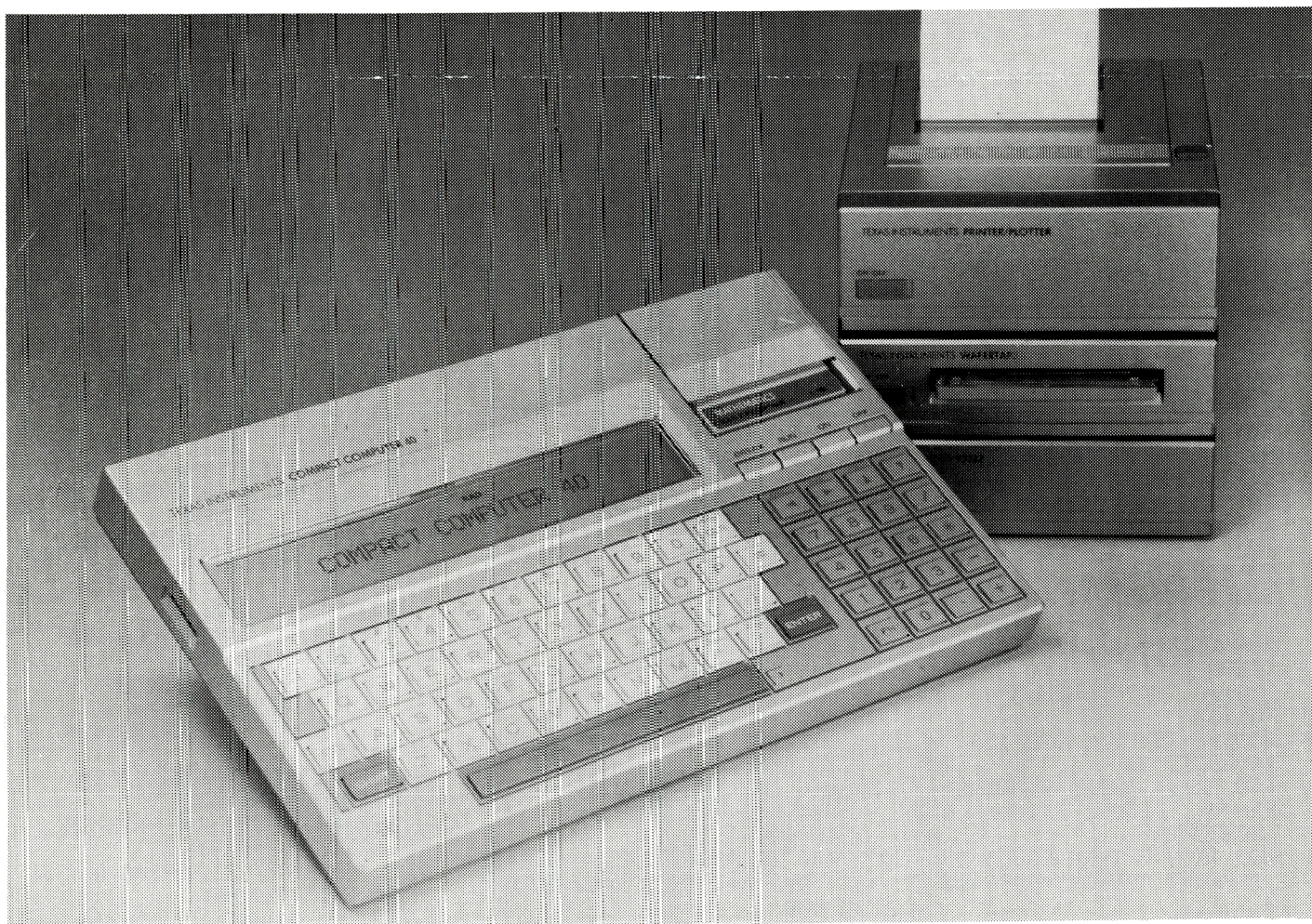
keying. A tilt stand is built into the back of the console to provide an optimum viewing and keying angle.

The unit is programmable in Enhanced BASIC, similar to that used by TI's family of home computers. BASIC and the operating system are provided in 34 kilobytes of built-in ROM. The unit comes standardly equipped with 6K of user RAM, expandable internally to 16K. An additional 128 kilobytes of ROM containing advanced professional and personal programs may be plugged into a port provided for that purpose.

The list price of the basic CC-40 is \$249.95. Deliveries are to start during the first quarter of 1983.

Three low-cost peripherals will be made immediately available to complement the computer. These are: an RS-232 interface, a printer/

*(continued on page 7)*





## EPSON DELIVERS MANUALS FOR THE HX-20

Epson is now delivering BASIC programming manuals for their HX-20 portable computer along with a couple of "bonus" program packages.

The two-volume BASIC manual consists of a 250 page "tutorial" for those who have not had previous programming experience and a second 250 page "reference" that is chock full of information to aid more experienced users.

The "bonus" packages consists of two cassettes containing software. The "Personal Entertainment" cassette has 10 "standards" such as Blackjack, Poker, Artillery, Hangman and Tic-Tac-Toe. The "Personal Productivity" cassette includes programs that provide a Clock, Event Timer, Memo Writer, and Ten Key Calculator among others. While all of the programs are relatively simple and exhibit signs of having been hastily prepared, most users will appreciate the good will gesture of at least providing some ready-to-run programs for the HX-20. A few samples of the output from these programs on the built-in HX-20 printer are provided here for your perusal.

For those looking for something else to load into a new HX-20, we are including our standard "benchmarking" Weather program for use and programming comparison purposes. This is an adaptation of the program originally supplied by *Emerich Auersbacher* for the Sharp PC-1500.

The Epson HX-20 has a lot of nice features and appears to be gaining a warm reception from the buying public. PCN plans to provide further reviews of the machine's capabilities in upcoming issues.

## Example Outputs from Epson HX-20 Printer



YOUR HANDS SHOWS:  
NOTHING  
YOU HAVE LOST  
50 DOLLARS  
YOU HAVE \$ 50

11:10 AM  
THU  
FEB  
4

This is a test of the  
Epson mini word  
processor. It allows  
you to input text and  
stuff, a line at a  
time, then sends it to  
the printer.

|   |   |   |         |   |
|---|---|---|---------|---|
| 0 | x | 0 | 0       | 0 |
| 0 | x | x |         |   |
| x | 0 | 0 | THE CAT |   |

### Program Epson HX-20 Weather Forecaster

```

10 DIM A$(9):CLS:PRINT "
  PRESS RETURN FOR":PRIN
T:PRINT " WEATHER FOREC
ASTER":
15 X$=INKEY$:IF X$="THE
N 15
20 A$(1)="S":A$(2)="SW":
A$(3)="W":A$(4)="NW":A$(
5)="N":A$(6)="NE":A$(7)=
"E":A$(8)="SE"
30 CLS:PRINT "BAROMETRIC
  PRESSURE":INPUT"(INCHES
)":P:IF P>30.2 THEN S=0
40 IF P<30.2 THEN S=1
50 IF P<30.1 THEN S=2
60 IF P<30.0 THEN S=3
70 IF P<29.8 THEN S=4
80 PRINT "(R)ISE, (F)ALL
, ":INPUT "OR (S)TEADY":
T$:X=0
90 IF T$="R"THEN PRINT "
  RISING (F)AST OR":INPUT
"RISING (S)LOW":R$:IF R$
="F" THEN X=1:GOTO 150
100 IF T$="R"THEN X=2:GO
TO 150
110 IF T$="F" THEN PRINT
  "FALLING (F)AST":INPUT
"OR (S)LOW":R$:IF R$="F"
  THEN X=3:GOTO 150
120 IF T$="F"THEN X=4:GO
TO 150
130 IF T$="S" GOTO 150
140 GOTO 80
150 PRINT "WIND BLOWING"
:INPUT "FROM THE":W$:CLS
:GOSUB 200
160 IF V=0 THEN SOUND 25
,1:PRINT "ERROR":GOTO 15
0
170 GOTO 300
200 V=0:FOR Z=1 TO 8
210 IF W$=A$(Z)THEN W=Z:
Z=8:V=1
220 NEXT Z:RETURN
300 V=ABS(W-3)<2
310 IF V*(S<3)ANDX=0 THE
N Z=1:GOTO 600
320 IF V*(S=1)ANDX=1 THE
N Z=6:GOTO 600
330 IF V*(S=0)ANDX=4 THE
N Z=3:GOTO 600
340 V=W=1 OR W=8
350 IF V*(S=1)ANDX=4 THE
N Z=4:GOTO 600
360 IF V*(S=1)ANDX=3 THE
N Z=5:GOTO 600
370 V=ABS(W-7)<2
380 IF V*(S=1)ANDX=4 THE
N Z=4:GOTO 600
390 IF V*(S=1)ANDX=3 THE
N Z=5:GOTO 600
400 IF V*(S>2)ANDX=4 THE
N Z=6:GOTO 600
410 IF V*(S>2)ANDX=3 THE
N Z=7:GOTO 600
420 V=W=6 OR W=7
430 IF V*(S<2)ANDX=4 THE
N Z=8:GOTO 600
440 IF V*(S<2)ANDX=3 THE
N Z=9:GOTO 600
450 IF (W=1 OR W=2) AND
S>2 AND X=2 THEN Z=10:GO
TO 600
460 IF (W<2 OR W>6) AND
S=4 AND X=3 THEN Z=11:GO
TO 600
470 IF ABS(W-6)<2 AND S=
4 AND X=3 THEN Z=12:GOTO
600
480 IF ABS(W-3)<2 AND S=
4 AND X=1 THEN Z=13:GOTO
600
490 IF X<3 AND ABS(W-5)<
3 THEN Z=2:GOTO 600
500 Z=1:IF X=3 THEN Z=8
600 PRINT "    FORECAST
-->":PRINT:K$=""
610 ON Z GOSUB 710,720,7
30,740,750,760,770,780,7
90,800,810,820,830
620 X$=INKEY$:IF X$="TH
EN 620 ELSE 30
710 PRINT "FAIR, LITTLE
CHANGE.":RETURN
720 PRINT "FAIR, LITTLE
COOLER.":RETURN
730 PRINT "    FAIR, WAR
MER.":RETURN
740 PRINT "RAIN WITHIN 2
4 HOURS":RETURN
750 PRINT "WINDY/RAIN IN
12 HRS":RETURN
760 PRINT "    CLOUDS/RAIN
COMING.":RETURN
770 PRINT "    RAIN & HIGH
WINDS.":RETURN
780 PRINT "UNSETTLED/MAY
BE RAIN":RETURN
790 PRINT"    RAIN/SNOW & W
INDY.":RETURN
800 PRINT "    CLEARING & F
AIRER."
810 PRINT "SEVERE STORM
WARNING":RETURN
820 PRINT "HEAVY RAIN/SN
OW/WIND":RETURN
830 PRINT "    CLEARING & C
OOLER.":RETURN

```

## A MACHINE-LANGUAGE MONITOR FOR THE PC-1500/PC-2

This Monitor for the Sharp PC-1500/Radio Shack PC-2 is designed to make it easier to write, load and debug machine-language programs. The program was created by *Norlin Rober*. Norlin is the originator of the mnemonics which *PCN* uses when discussing these units' machine-language. (The instruction set was presented in the Special Edition of *PCN* published in September of 1982. A summary of the mnemonics and machine codes, on a pocket-size card for easy reference, is being distributed with this issue of *PCN*.)

This is what is sometimes referred to as a "hybridized" program. The portion of the program involving operator inputs and outputs is written in BASIC. The remainder is in machine language. One reason for using the latter is that execution is much faster, particularly for operations such as MOVE and HUNT.

The Monitor includes the commands shown in the accompanying list. They are executed by the indicated user-defined keys.

### List Monitor Commands

|       |                                                                                                                                        |
|-------|----------------------------------------------------------------------------------------------------------------------------------------|
| DEF/S | STORE codes into memory.                                                                                                               |
| DEF/V | VIEW memory contents.                                                                                                                  |
| DEF/L | LIST memory contents using printer.                                                                                                    |
| DEF/H | HUNT for a specified byte.                                                                                                             |
| DEF/M | MOVE a block of stored codes.                                                                                                          |
| DEF/B | BREAKPOINT insertion in user's m.l. program.                                                                                           |
| DEF/J | JUMP to specified address in user's m.l. program.<br>(If a breakpoint is reached, contents of CPU registers and stack may be printed.) |
| DEF/A | ALTER contents of CPU registers at breakpoint.                                                                                         |
| DEF/C | CONTINUE execution following breakpoint.                                                                                               |
| DEF/F | FIX program (remove breakpoint).                                                                                                       |
| DEF/X | EXIT from STORE, VIEW or HUNT mode.                                                                                                    |
| DEF/= | Convert decimal number in display to hexadecimal.                                                                                      |

### Loading the Monitor into Memory

This program is intended for use in a PC having an 8K memory module. The first step in loading the program is to execute the command: NEW &4000. This reserves a portion of memory for machine-language purposes. There are then two parts to entering the actual Monitor:

- 1). The accompanying machine-language codes must be entered into an appropriate section of memory using POKE statements. Although this is a tedious process, once it is finished the codes may be stored on cassette for easy loading in the future. It is important that the codes be stored in the exact locations specified.

- 2). The accompanying BASIC portion of the program is then keyed into memory in the usual way. It too may be saved on cassette for ease in future loading of the Monitor.

It is a good idea to save the program on cassette *before* testing it! Then, in the event you have made any keying errors that result in a bad program crash, you will not have to key everything all over again.

### Saving/Loading the Monitor Program Using a Cassette

After the two parts of the program have been keyed into memory, they should be recorded on cassette as follows:

- 1). Execute CSAVE M "MONITOR PART 1"; &3E4F, &3FFF
- 2). Execute CSAVE "MONITOR PART 2"

To reload the program from cassette, use:

- 1). NEW &4000
- 2). CLOAD M
- 3). CLOAD

### Monitor Memory Map

The Monitor program uses memory areas as follows:

3800 — 38C4 REServe memory area, not used by Monitor.

38C5 — 3DFF Available for user's m.l. routines (1339 bytes).

3E00 — 3E4B Monitor temporary registers and stack area.

3E4F — 3FFF Monitor machine-language routines.

4000 — 5FFF BASIC storage area. Monitor uses 929 bytes.

The Monitor also uses variables A, B, C, A\$, B\$, C\$ and D\$. Thus, those memory areas (&7900 — &7917 and &78C0 — &78FF) may be altered when the Monitor is in operation.

### Operating the Monitor

Place the PC in the RUN mode and use the DEFined keys to select the various operations. Always enter addresses as four hexadecimal digits (using 0 — 9 and A — F). Machine codes should be entered as two-digit hexadecimal numbers. Note that (for example) the code 0B should be entered by keying both 0 and B, not simply B. The & prefix normally used to indicate hexadecimal values to the PC is not necessary when responding to Monitor prompts.

The following discussion provides information on the use of each of the Monitor commands:

**STORE:** DEF/S places the computer in the mode used to load and store machine code. The Monitor will prompt for the beginning address at which codes are to be stored. Respond to this prompt with a four-digit hexadecimal address (such as 38C5) and then press the ENTER key. The program will then show the address into which the next byte of code will be placed. The code must be entered as a two-digit hexadecimal number (for example, B5) which is terminated by the ENTER key. After each byte has been accepted, the display shows the next sequential address. You may thus continue entering object code into successive memory locations. If you want to skip over an address, then just press ENTER without any input. This advances the Monitor to the next address, leaving the contents of the skipped address unchanged. To exit this mode, key DEF/X.

**VIEW:** Key DEF/V to place the Monitor in the mode used to examine the contents of memory. Enter a beginning address when prompted. The computer will then display the contents of memory in groups of four bytes. Press the ENTER key to advance to the next group of four bytes. Use DEF/X to terminate this mode.

**LIST:** Initiated by DEF/L, this command operates similarly to VIEW, except that the memory contents are printed instead of being displayed on the LCD. However, two address prompts will appear: one to indicate the starting address, the other denoting the ending location.

**HUNT:** Key DEF/H. Enter the byte being sought at the prompt. A second prompt asks for the address at which the Monitor is to begin searching. When a byte having the indicated value is located, the display will show the address, the byte itself, and the three bytes that follow it. Press ENTER to continue the search for another occurrence of the same byte.

As an example, the Monitor command makes it easy to locate the codes used at the beginning of BASIC program lines. Simply HUNT for the byte 0D! (The code 0D is the ASCII representation for "carriage return" used to terminate lines of BASIC.) Given 4000 as a beginning address (assuming this Monitor program is in memory), the display will show 401E 0D 00 0C 08. This indicates that the first BASIC program line (of the Monitor) terminates at address 401E. The next line number (using two bytes starting at 401F) is 00 0C (line 12 in decimal). The last byte (08) in the display represents the "link" byte. It indicates the number of bytes that remain in that BASIC program line. If you press the ENTER key, you will find where the next line ends. Use DEF/X to exit from this mode.

The HUNT command is also highly useful as an aid to locating specific instruction codes when investigating ROM.

The speed at which HUNT operates dramatically illustrates the advantage of using machine language.

**MOVE:** DEF/M initiates this mode. Using it, a block of stored codes of any size may be moved, in either direction, to any specified RAM location. The new location may overlap the original. Unless there is overlapping, the original block of code remains unchanged by the operation.

Prompts request the beginning and ending addresses of the original block of memory, and then the beginning address of the location to

which it is to be transferred.

Be careful when using this command. If an incorrect address is inadvertently given, you may destroy parts of memory that you did not want altered! A wrong move operation can alter the Monitor program itself or disturb pointers in system RAM, possibly resulting in a system crash.

The following example may be used to familiarize yourself with the command. The variable J is stored (by BASIC) in RAM addresses 7948 to 794F. Variable M begins at address 7960. You can copy the contents of variable J into M using MOVE. Use BASIC to store some arbitrary test number in J. Key DEF/M to initiate MOVE. Enter the address 7948 in response to the prompt BEGIN BLOCK and address 794F for END BLOCK. In response to MOVE TO, key in the address 7960. Press ENTER at the end of this address and the MOVE command will be executed. You can then use BASIC to display the contents of the variable M. It should contain the same value as that originally placed in J.

**BREAKPOINT:** This command is useful in debugging machine-language programs. What it does is remove three bytes from a program in RAM, at whatever address is specified, and save them elsewhere. The three locations are then loaded with opcodes for an instruction to jump to a register-saving routine, which saves the contents of the CPU registers, flag status register and the stack. If execution is resumed later, (for example, by the Monitor's CONTINUE command), the CPU registers, flags and stack will be restored.

Note that the BREAKPOINT command executed by DEF/B simply inserts an instruction to jump to the register-saving routine. It does not execute the user's machine-language program. Also note that only one breakpoint should be in a program at any given time.

As a note of caution: be sure that the address entered for insertion of a breakpoint is the address of the *first byte* of a machine-language instruction!

**JUMP:** Key DEF/J to use this routine. The address entered in response to the prompt will be taken as the beginning address of the code to be executed.

This command may be used in lieu of a BASIC "CALL" instruction to begin executing a machine-language program. It *must* be used when debugging a program containing a breakpoint.

If a program containing a breakpoint is executed (using the JUMP command), BREAKPOINT REACHED will be displayed when the breakpoint is found. The user may then initiate a printout of the CPU registers, flags and stack by pressing ENTER.

**ALTER:** DEF/A will begin the routine for altering the contents of the CPU registers following the reaching of a breakpoint. When the prompt for a particular CPU register appears, the new value should be entered. Note that CPU register A contains one byte, requiring inputting of two hexadecimal digits. Registers X, Y and U, on the other hand, each hold two bytes, hence four-digit values should be entered for those registers.

If you want to leave a particular register undisturbed, press ENTER alone when the prompt for that register appears.

After all desired alterations have been entered, the three bytes that had been removed from the program by the BREAKPOINT command will be replaced. Execution of the machine-language program then continues from where it left off, with the CPU registers now containing the altered values specified.

**CONTINUE:** Execution of DEF/C will continue execution of a program following a breakpoint, with no alteration of the CPU registers.

**FIX:** The DEF/F command replaces the three bytes that were removed by a breakpoint directive. It is only needed if neither ALTER or CONTINUE are used following establishment of a breakpoint, since those two commands make the same replacement.

**EXIT:** Use DEF/X to exit from the STORE, VIEW or HUNT modes. The command is provided as a convenience, rather than a necessity. (The use of SHIFT/CL would accomplish the same effect.)

**HEXADECIMAL** equivalents of calculated results may be obtained using DEF/= . This is useful for such things as calculating the required size of a relative branch. An example illustrating hexadecimal conversion follows:

Enter 35 \* 61 into the display. Now key DEF/= . The product will

be shown as 0857 which is the hexadecimal equivalent of the product of the decimal numbers 35 and 61. Enter &3A56 - &39EB and key DEF/= . You will have an instant answer for a hexadecimal subtraction!

You may also simply convert a decimal number to hexadecimal by entering the number into the display and using DEF/= .

#### The BREAKPOINT Illustrated

To amplify how BREAKPOINT and related commands operate, I will illustrate their use with a specific example.

To begin, input a short, simple machine-language program. Put the codes into memory using the STORE command. Here is such a routine:

| Address | Opcodes  | Mnemonics |
|---------|----------|-----------|
| 38C5    | 48 12    | LDXH #12  |
| 38C7    | 4A 34    | LDXL #34  |
| 38C9    | 58 56    | LDYH #56  |
| 38CB    | 5A 78    | LDYL #78  |
| 38CD    | 68 9A    | LDUH #9A  |
| 38CF    | 6A BC    | LDUL #BC  |
| 38D1    | B5 00    | LDA #00   |
| 38D3    | FD 88    | PSHX      |
| 38D5    | FD 98    | PSHY      |
| 38D7    | FB       | SETC      |
| 38D8    | FD 1A    | POPY      |
| 38DA    | FD 0A    | POPX      |
| 38DC    | AE 39 00 | STA 3900  |
| 38DF    | A4       | LDA UH    |
| 38E0    | AE 39 01 | STA 3901  |
| 38E3    | 24       | LDA UL    |
| 38E4    | AE 39 02 | STA 3902  |
| 38E7    | 9A       | RTS       |

A careful examination of this example routine will reveal that it does not accomplish anything of any practical importance. The purpose of the routine is simply to illustrate how some of the Monitor features perform.

After storing the example codes, use VIEW to make certain that they have been entered correctly. Then try executing the program with the JUMP command, giving address 38C5 as the response to the BEGIN prompt. To see whether the program did what it was designed to do, use VIEW again to see if addresses 3900 to 3903 contain appropriate values.

Try a breakpoint at the address 38D8 by keying DEF/B and entering 38D8 at the prompt. Then execute JUMP again using 38C5 as the starting location. The BREAKPOINT REACHED message should soon appear. Assuming that your PC is connected to its printer, press the ENTER key. Compare the printout against what the machine-language routine should have accomplished to that point. (The "CIZ" in the printout means that flags C, I and Z were set at the time the breakpoint was encountered.)

You can then alter the contents of, say, registers A and U, then see whether the altered values eventually find their way to addresses 3900 to 3902. First, key DEF/A. The prompt "A:" appears to request a new value for register A. Enter, for trial purposes, "BB" and press the ENTER key. Skip over changing X and Y by just pressing ENTER when their prompts come up. When "U:" appears, input FFEE as a new value and then press ENTER.

Use VIEW starting at address 3900 to verify that the CPU alterations took effect.

Key DEF/F to repair the breakpoint. You will be informed that there is no breakpoint to be fixed! Remember, using the ALTER feature automatically "fixes" the breakpoint. If you try either ALTER or CONTINUE at this point (without putting in another breakpoint), you will get the same response.

However, there is no built-in protection against your entering a second breakpoint without having fixed the first one. If you put in more than one breakpoint at a time, the FIX routine will only correct the last one entered. (The Monitor will have then lost track of where you placed the first breakpoint and what the original bytes were.)



# Program Machine Language Portion of Monitor

```

3E4F BE 3F DE 58 3F2B 46 9A BE 3F
3E53 3F 5A 19 84 3F2F DE FD 5A F4
3E57 51 04 51 F5 3F33 79 05 FD 28
3E5B F5 F5 5B FF 3F37 FB A5 79 0E
3E5F 46 B5 71 43 3F3B 00 2A A5 79
3E63 B5 3E 43 B5 3F3F 0D 80 28 84
3E67 BA 0E 9A EB 3F43 96 81 0E 89
3E6B 3E 70 FF FD 3F47 04 04 16 81
3E6F 5E FF FD C8 3F4B 08 F5 88 03
3E73 FD 88 FD 98 3F4F FD 62 93 07
3E77 FD A8 FD AA 3F53 9A 24 FD CA
3E7B FD C8 FD 48 3F57 FD DA 84 A2
3E7F FD 88 46 58 3F5B 08 94 A2 18
3E83 3E 5A 00 F5 3F5F 47 53 88 04
3E87 4E 49 91 05 3F63 FD 62 93 08
3E8B AA 78 48 E9 3F67 9A F2 48 79
3E8F 3E 70 00 9A 3F6B 4A 10 BE DC
3E93 48 3F 4A 19 3F6F 20 58 78 5A
3E97 BE 3F 9A 5A 3F73 E0 48 7A CD
3E9B D0 48 3E 4A 3F77 6C B7 00 8B
3E9F AF A5 3E 02 3F7B 02 B5 2D BB
3EA3 6A 04 D5 81 3F7F 20 51 D0 00
3EA7 02 F5 46 44 3F83 04 FD 28 8E
3EAB 88 08 8E DE 3F87 32 56 CD 2A
3EAF 43 49 5A 56 3F8B 65 05 8E 44
3EB3 48 48 3E 4A 3F8F BE 3F B8 6A
3EB7 03 BE 3F 9A 3F93 03 BE 3F AE
3EBB 5A F0 BE 3F 3F97 88 05 9A 5A
3EBF 9C 5A C0 BE 3F9B E0 58 78 B5
3EC3 3F 9C 8E E3 3F9F 20 51 8E 02
3EC7 48 3F 4A 19 3FA3 5A E0 BE 3F
3ECB 45 18 45 1A 3FA7 B5 8E 0B 58
3ECF F5 F5 F5 49 3FAB 78 5A D0 B5
3ED3 00 9A 48 3E 3FAF 20 51 8E 02
3ED7 4A 09 8E FC 3FB3 5A D0 45 8E
3EDB 4A 07 8E 06 3FB7 07 5A C0 84
3EDF 4A 05 8E 02 3FBB BE 3F BF 04
3EE3 4A 03 48 3E 3FBF 58 78 28 F1
3EE7 BE 3F D7 8E 3FC3 BE 3F C7 A4
3EEB ED AA 3D FF 3FC7 F9 B9 0F B3
3EEF FD 0A FD 4E 3FCB 30 B7 3A 81
3EF3 44 58 3E 5A 3FCF 02 B3 06 51
3EF7 02 55 41 4E 3FD3 59 00 FB 9A
3EFB 49 91 06 FD 3FD7 5A D0 BE 3F
3EFF 8A FD EC FD 3FDB E9 41 9A 5A
3F03 2A FD 1A FD 3FDF C0 BE 3F E9
3F07 0A 8B 08 FD 3FE3 08 BE 3F E9
3F0B 8A ED C0 00 3FE7 0A 9A 58 78
3F0F FF 8E 06 FD 3FEB BE 3F F6 F1
3F13 8A ED C0 00 3FEF 28 BE 3F F6
3F17 00 BA 00 00 3FF3 A2 FB 9A 55
3F1B 00 00 00 00 3FF7 B7 40 81 02
3F1F A5 79 15 F7 3FFB B3 08 B9 0F
3F23 99 03 46 BE 3FFF 9A
3F27 3F 8F 46 46

```

## Crash Recovery

Anyone experimenting with machine language soon learns that the PC (or any computer!) is totally unforgiving of errors. There are none of

# Program BASIC Portion of Monitor

```

10: "="AREAD C: INT REACHED"
CALL &3F68: 46:CALL &3E93:
CURSOR 21: LPRINT "ADDRES
PRINT C$:END S: ";C$:LPRINT
12: "S"GOSUB 68 "FLAGS: ";B$
14: CALL &3FB8,A: 48:CALL &3EB4:
WAIT 0:PRINT A LPRINT "A: ";B$
$:WAIT :INPUT :LPRINT "X: ";A
": ";B$:CLS : $:LPRINT "Y: ";
CALL &3FD7,A: D$:LPRINT "U: "
GOTO 14 ;C$
16: CLS :A=A+1: 50:B=64-PEEK &3E0
GOTO 14 1:IF B>0LPRINT
18: "V"GOSUB 68 "STACK: ";A=&3
20: CALL &3F8F,A: E0A:FOR C=1TO
PRINT A$:GOTO B:CALL &3FAA,A
20 :LPRINT B$;;
22: "L"GOSUB 72 NEXT C:LPRINT
24: CALL &3F8F,A: 52:LPRINT :END
LPRINT A$:IF A 54: "A"GOSUB 76:
>BEND INPUT "A: ";B$
26: GOTO 24 :CALL &3ED5
28: "H"CLS :INPUT 56: INPUT "X: ";B$
"BYTE SOUGHT: :CALL &3EDB
":A$:CALL &3FD 58: INPUT "Y: ";B$
E,C:GOSUB 68 :CALL &3EDF
30: CALL &3F1F,A: 60: INPUT "U: ";B$
PRINT A$:GOTO :CALL &3EE3
30 62:CALL &3EEC:END
32: "M"CLS :INPUT 64: "C"GOSUB 76:
"BEGIN BLOCK: GOTO 62
":A$:CALL &3FD 66: "F"GOSUB 76:
E,A:INPUT "END END
BLOCK: ";A$: 68:CLS :INPUT "BE
CALL &3FDE,B GIN: ";A$:CALL
34: INPUT "MOVE TO &3FDE,A:RETURN
: ";A$:CALL &3 70:GOTO 68
F2D:END 72:GOSUB 68:INPUT
36: "X"CLS :END "END: ";A$:
38: "B"CLS :INPUT CALL &3FDE,B:
"BREAKPOINT: " RETURN
:A$:CALL &3E4F 74:GOTO 72
:END 76:CLS :IF PEEK &
40:GOTO 38 3F1ECALL &3EC7
42: "J"GOSUB 68: :RETURN
CALL &3E6A,A: 78:PRINT "NO BREA
IF PEEK &3E70 KPOINT":END
END STATUS 1 929
44:PRINT "BREAKPO

```

the friendly "ERROR" messages as found in BASIC. In many cases, the simplest mistakes will be punished by "crashes" which result in the PC becoming completely locked up. The only thing to be done in such situations is to use the "All Reset" key on the back of the PC. But, all is not necessarily always lost!

Instead of following the instructions given in the manual when a crash occurs, try using the following approach. Nine times out of ten your program will still be intact: First, do *not* hold in the ON key while pressing "All Reset." Secondly, after you obtain "NEW0? CHECK" on the display, do *not* execute NEW0! Instead, key SHIFT

and then CL (SHIFT/CL). Then, survey the damage, if any, to the contents of memory. If you find things messed up beyond repair, then you will have to key in NEW0 and start over. Often, however, you will find memory contents retained with little alteration.

The Monitor program does contain a few safeguards. If you neglect to key in an address before pressing ENTER, the computer will repeat the prompt. If you forget any of the three addresses required for a MOVE, it will restart the entire MOVE routine from the beginning. And, if you use the hexadecimal conversion routine with a result exceeding FFFF (or less than -FFFF), the Monitor will put the word ERROR into the display. Otherwise, you are on your own!

#### Combined Monitor/Disassembler

The use of this Monitor program can be greatly enhanced by coupling it with the Disassembler program published in PCN (Special Edition accompanying Issue 17). Those who have the Disassembler on a cassette can simply MERGE that program with this Monitor. To facilitate joint use of the Monitor and Disassembler, change the Disassembler as follows:

- 1). Insert "D" as a label at the beginning of line 10.
- 2). Delete lines 90, 91 and 92 as they are no longer needed.
- 3). It would also be advisable to eliminate lines 12, 80 and 82.

Now DEF/D may be used to DISASSEMBLE! A printed list of mnemonics generated by the Disassembler is often of considerable help in finding elusive machine-language programming errors!

If you are interested in doing machine-language programming on your PC-2/PC-1500, you should find this Monitor of considerable value. The 1339 bytes (from address &38C5 through &3DFF) that are available for machine-language routines should be adequate for most purposes. (If you are able to write longer routines, you should have little difficulty figuring out how to relocate the Monitor program!)

#### SIDELISTER

Mel Beckman, 717 West Broadway, Winona, MN 55987, developed this program and the following narrative:

One inconvenience of pocket computers that I had resigned to put up with is the hard to read program listings produced on those teeny weeny printers. Recently it occurred to me that Sharp had nearly provided an alternative to this in the form of sideways printing. The ability to print sideways going up, down or inverted is provided by the ROTATE command. ROTATE 0 causes text to print normally. Using ROTATE 1 turns the output 90 degrees clockwise; ROTATE 2, 180 degrees; and ROTATE 3, 270 degrees. After a ROTATE has been executed, all subsequent LPRINT directives will print in that orientation. You might expect that entering ROTATE 1 followed by LLIST would cause the program listing to print sideways. The nice thing about such a listing would be the ability to print long lines without those eye-jogging continuations that occur when long lines have to be printed down a column. Alas, it is not that simple.

#### Where There Is a Will. . .

A somewhat clumsy method of obtaining a sideways program listing is to write a small basic program that PEEKs at the entire program and prints it using ROTATE and LPRINT. That is what this program does. It is really not an ideal solution to the problem, but it does produce a sideways listing. The program is designed to be appended to the end of a user's program. Thus, it uses high line numbers (64000+) to avoid conflict with the user's code. The output may be printed in either CSIZE 1 or CSIZE 2. The former permits up to 84 characters per line and is nice for archival listings. CSIZE 2 prints up to 42 characters per line and is good for debugging sessions. The program automatically advances to a new "page" (after 20 lines at CSIZE 1 or 10 lines at CSIZE 2). It also correctly continues unusually long statements onto the next line when necessary.

#### Program Sidelister

|                       |                        |                      |
|-----------------------|------------------------|----------------------|
| 64000: "L"REM SIDE    | C: IF M>(84/Z          | PEEK (U+N+2)         |
| LISTER                | )LET M=84/Z            | =W)GOTO 6449         |
| 64025: Z=2: X=212-Z*  | 64205: C=0: X=X-Z*10   | 0                    |
| 6                     | : GLCURSOR (X          | 64475: U=U+N+5: IF U |
| 64050: GRAPH :        | , 0)                   | >&C348LET U=         |
| GLCURSOR (X,          | 64210: L=L+1: IF L=    | &B054: GOTO 6        |
| 0): SORGN :           | INT (20/Z)             | 4460                 |
| ROTATE 1:             | GOSUB 64300            | 64480: IF (U<&C053)  |
| CSIZE Z               | 64215: RETURN          | AND (U>&B0E2         |
| 64100: X=0: L=0: C=0: | 64300: L=0: X=0: Y=- ( | )LPRINT "? "         |
| M=0: Q=STATUS         | Z*M*6.4):              | : Q=Q+2: C=C+        |
| 2-STATUS 1:           | GLCURSOR (X,           | 2: RETURN            |
| LPRINT "*TOP          | Y): SORGN : M=         | 64485: GOTO 64460    |
| *"; TIME              | 0: RETURN              | 64490: IF (C+N+1)>(  |
| 64105: GOSUB 64200    | 64400: IF PEEK Q>&E    | 84/Z)GOSUB 6         |
| 64110: IF PEEK (Q)=   | 5GOTO 64450            | 4500                 |
| &FFEND                | 64405: IF C>(84/Z)     | 64495: FOR J=1 TO N: |
| 64115: LPRINT USING   | GOSUB 64500            | LPRINT CHR\$         |
| "#####";              | 64410: LPRINT CHR\$    | (PEEK (U+J))         |
| PEEK (Q)*256          | PEEK Q: C=C+1          | : NEXT J             |
| +PEEK (Q+1);          | : Q=Q+1;               | 64498: LPRINT " "; : |
| ": "                  | RETURN                 | C=C+N+1: Q=Q+        |
| 64120: Q=Q+3: C=C+7   | 64450: U=PEEK (Q): W   | 2: RETURN            |
| 64125: IF PEEK (Q)=   | =PEEK (Q+1):           | 64500: GOSUB 64200:  |
| &0DLET Q=Q+1          | U=&C053                | LPRINT "             |
| : GOTO 64105          | 64460: N=PEEK UAND     | "; : C=C+7:          |
| 64130: GOSUB 64400    | &0F                    | RETURN               |
| 64135: GOTO 64125     | 64470: IF (PEEK (U+    | STATUS 1 739         |
| 64200: IF C>MLET M=   | N+1)=U)AND (           |                      |



## How It Works

The first task of Sidelister is to determine where the program it is going to list starts in memory. STATUS 2 less STATUS 1 provides that value. Sidelister thus starts PEEKing at that address and stops when it finds a hexadecimal value of FF as that is the value used to signify the end of a program.

BASIC statements are not actually stored character-for-character in the PC-1500/PC-2. To conserve space the line number is saved as a two-byte binary value and keywords are converted into two-byte "tokens". The format of such a compressed BASIC statement line is:

|               |             |            |     |
|---------------|-------------|------------|-----|
| 2-byte line # | length byte | statements | &0D |
|---------------|-------------|------------|-----|

The text length byte shown in the diagram is used by the BASIC interpreter to enable it to rapidly scan forward through a program. Since the Sidelister program will be processing every program line, it can ignore this byte. The code &0D at the end of every program line represents an ASCII carriage return. It is used by Sidelister to advance to the next line to be printed.

Tokens are expanded back into keywords using the keyword/token table stored in ROM. This is the same table that the BASIC interpreter uses. There are actually two tables. One starts at address &C053, the other at &B054 (for the CE-150 printer/cassette statements). Each table entry is formatted as follows:

|             |         |            |         |
|-------------|---------|------------|---------|
| length byte | keyword | token code | address |
|-------------|---------|------------|---------|

The first byte of the table entry contains the length of the keyword in the four low-order bits. The high-order bits are apparently used as some sort of flag. Extracting the four low-order bits is accomplished by using an AND with the value &0F. The expanded keyword appears next in the table, followed by the two-byte token value used to represent the keyword. Tokens always begin with a byte greater than &E5 which makes them easy to identify. The last two bytes of each entry contain the address of the ROM routine that processes the statement.

A simple BASIC loop is used to scan these tables and translate the tokens into corresponding keywords for the listing. A blank is added to the end of each keyword for spacing.

Referring to the Sidelister program listing, lines 64000 — 64100 perform "housekeeping". Note variable Z in line 64025. Set this to select the CSIZE value (1 or 2) desired. While values of 3 through 9 could conceivably be used, the listings would be unduly large. Lines 64105 — 64135 constitute the main program loop. It extracts and prints the line number, checks for a carriage return code, and terminates upon encountering an &FF byte. The "new line" routine begins at line 64200, the "new page" at 64300, and the print routine at line 64400. The print routine fetches a byte, determines if it is a token that needs expansion, expands it if required, then prints appropriately. Line 64500 is used to continue long statements onto the next line.

## Is It Practical?

After comparing various program listings in vertical and sideways formats, I strongly prefer sidelistings. The elimination of a lot of continuation lines improves readability. Debugging a program from a sideways listing is less strenuous on the eyes. The logical flow of a program seems more apparent. It is nice to be able to fold a listing between "pages" thereby eliminating annoying creases in the middle of a line. The horizontal arrangement is easier to scan than a skinny vertical printout. I would use this listing format exclusively if it could be obtained without undue effort.

Earlier I stated that this was not the ideal solution to the sidelisting problem. The best solution would be to have the capability provided in ROM. Maybe some future version of the unit will include this feature. The program suffers from a number of disadvantages that make practical application doubtful in its present form. Not the least of these is its slow execution speed. Looking up tokens with a BASIC search routine is not quick (taking 1 — 8 seconds each). Another drawback is having to

append the program at the end of each program that is to be so listed. This burden can be eased by merging the routine from tape, but the code still utilizes valuable memory space.

However, I did not write Sidelister as a practical answer to the problem. It was an experiment; a tool to investigate the usefulness of such listings. In this capacity it has served well. The program is functional, but not efficient. It was developed using the simplest methods available without regard to execution speed or memory usage. This is a viable way to approach many software projects: programs can always be "tuned up" later; get them running first!

Immediate improvements readily come to mind. Somewhere in ROM is a routine that will return a keyword when given a token as input. Finding this routine would considerably reduce the execution time and cut the program size perhaps in half. There may well be a ROM routine that returns expanded statements, line-by-line. If not, machine-language routines could accomplish tremendous speed enhancements. I do hope that other ingenious persons will take the outline presented here and produce a Son of Sidelister. Perhaps additional features, such as the ability to print just selected lines or skip to a new page could be added.

If nothing else, perhaps the information disclosed about the internal storage of the token tables and BASIC program storage may inspire others to create additional programming tools!

## Example Output from Sidelister Program Using CSIZE 1

```
*TOP* 81814.5088
64000: "L"REM SIDELISTER
64025: Z=1: X=212-Z*6
64050: GRAPH : GLCURSOR (X,0): SORGN : ROTATE 1: CSIZE 2
64100: X=0: L=0: C=0: M=0: STATUS 2-STATUS 1: LPRINT "TOP*"; TIME
64105: GOSUB 64200
64110: IF PEEK (Q)=&FFEND
64115: LPRINT USING "#####"; PEEK (Q)*256+PEEK (Q+1); "; ";
64120: Q=Q+3: C=C+7
64125: IF PEEK (Q)=&0DLET Q=Q+1: GOTO 64105
64130: GOSUB 64400
64135: GOTO 64125
64200: IF C>MLET M=C: IF M<(84/Z)LET M=M*Z
64205: C=0: X=X-Z*10: GLCURSOR (X,0)
64210: L=L+1: IF L=INT (20/Z)GOSUB 64300
64215: RETURN
64300: L=0: X=0: Y=-(Z*M*6.4): GLCURSOR (X,Y): SORGN : M=0: RETURN
64400: IF PEEK (Q)>&E5GOTO 64450
64405: IF C<(84/Z)GOSUB 64500
64410: LPRINT CHR$ PEEK (Q): C=C+1: Q=Q+1: RETURN
64450: U=PEEK (Q): W=PEEK (Q+1): U=&C053
64460: N=PEEK UAND &0F
64470: IF (PEEK (U+N+1)=U)AND (PEEK (U+N+2)=W)GOTO 64480
64475: U=U+N+5: IF U>&C348LET U=&B054: GOTO 64460
64480: IF (U<&C053)AND (U>&B0E2)LPRINT "? "; Q=Q+2: C=C+2: RETURN
64485: GOTO 64460
64490: IF (C+N+1)>>(84/Z)GOSUB 64500
64495: FOR J=1 TO N: LPRINT CHR$ (PEEK (U+J)); NEXT J
64498: LPRINT " "; C=C+N+1: Q=Q+2: RETURN
64500: GOSUB 64200: LPRINT " "; C=C+7: RETURN
```

## TI CC-40 (concluded from page 1)

plotter and a Wafertape digital tape drive. Other add-ons, scheduled to be available by the third quarter of 1983 include: a wand input device, modems, printers and a black and white TV monitor. These devices are also said to operate with the new TI-99/2 Basic Computer, and, using an appropriate adapter, will work with the TI-99/4A Family Computer.

List prices of the three immediately-available peripherals are: \$99.95 for the RS-232 interface, \$199.95 for the 4-color printer/plotter, and \$139.95 for the Wafertape drive.

The Wafertape drive will be capable of transferring data in digital format at approximately 1 kilobyte per second. This is considerably faster than most other portable systems now on the market.

Texas Instruments has also announced an array of 22 software application packages as being immediately available, with another 53 packages planned for introduction by the third quarter of 1983. Some software will be supplied on Wafertapes at \$19.95 each. These include: Elementary Dynamics, Regression/Curve Fitting, Pipe Design, Production and Planning, Inventory Control, Electrical Engineering, Thermodynamics, Photography and more. Other software packages will come in plug-in ROM cartridges. These include: Mathematics, Finance, Perspective Drawing, Statistics, Business Graphics, Nonparametric statistics, and Advanced Electrical Engineering (at \$59.95 each); an Editor/Assembler (\$124.95) and two games packages (at \$39.95 each).

## THE SHARP PC-1251

While Sharp Electronics is showing off the PC-1250 to U.S. customers, (see page 1, Issue 21 of *PCN*), the rest of the world is apparently being treated to the model PC-1251. What is the difference between the two models? Just this: the 1251 leaves the user about 3,700 bytes of memory for program and data storage, while the 1250 provides a mere 1,400. Otherwise, the two units are identical, separated only by approximately \$50.00 in price.

Does the extra memory make a big difference? You bet it does! The 3,700 bytes in the 1251 gives room to utilize the two-dimensional arrays that can be defined on both it and the 1250. The extra memory also permits about 2-1/2 times as many programs, routines or data to be instantly available!

During the past few weeks *PCN* has had the opportunity to try out a PC-1251. It does appear to be upwards compatible with the original PC-1211. In particular, the designers have handled the arrays properly (something they failed to do in the PC-1500) so that variables A — Z may also be referenced as array elements A(1) — A(26). This array can then be extended using additional elements starting with A(27), etc. Additionally, you can get away with using implied multiplication (with AB being interpreted as A times B) since the machine only provides for single character variable names. (Now, however, you can also have similarly named array elements, such as B(0), B(1), B(2), . . . , etc.) The upshot of this is that you can simply load in programs originally created for the PC-1211 without alteration and have them execute 5 to 6 times as fast! Previous PC-1211 owners should be delighted. Of course, you can then expand those programs to take advantage of some of the enhanced capabilities of the 1251. Perhaps most notable amongst the enhancements are the DATA, READ and RESTORE statements that so many people complained about missing on the earlier PC-1211.

Another nice feature of the 1251 is that it really does fit in a pocket, even a shirt pocket, quite comfortably. The keys are tiny, but spaced far enough apart so as to permit relative ease in the inputting of data or programs. The numeric keys are larger than the alphabetical buttons and are arranged in the traditional calculator format to facilitate rapid keying of numeric information.

For *PCN* aficionados, at least, the PC-1251 at a list price of \$159.95 would seem to be a better deal than the PC-1250, even though the 1250 has a list price of just \$109.95. It is too bad that the 1251 is not currently sold in the U.S. Could Sharp be persuaded to change their minds?

### The POCKET COMPUTER NEWSLETTER is Available!

*By Subscription Only:* for a calendar year period (January — December). You get all issues published to date for the calendar year in which you subscribe, at the time you subscribe.

MC/VISA Phone Subscriptions: (203) 888-1946

- ☐ 1982 Regular Subscriber (Issues 11 — 20). \$30.00 in U.S. (U.S. \$36.00 to Canada. U.S. \$45.00 elsewhere.)
- ☐ 1982/83 Subscriber (Issues 11 — 30). \$60.00 in U.S. (U.S. \$72.00 to Canada. U.S. \$90.00 elsewhere.)
- ☐ 1983 Regular Subscriber (Issues 21 — 30). \$36.00 in U.S. (U.S. \$42.00 to Canada. U.S. \$50.00 elsewhere.)

Orders must be accompanied by payment in full. We do not issue invoices for the POCKET COMPUTER NEWSLETTER.

Thank you for your remittance.

Name: \_\_\_\_\_  
Addr: \_\_\_\_\_  
City: \_\_\_\_\_ State: \_\_\_\_\_ Zip: \_\_\_\_\_  
MC/VISA #: \_\_\_\_\_ Expires: \_\_\_\_\_  
Signature: \_\_\_\_\_

## FROM THE WATCH POCKET

If you are having any problems with the Histogram program published in Issue 21 of *PCN*, then chances are you have a version A04 ROM. Drop *Russ Doughty* a SASE (self-addressed, stamped envelope) if you would like a new listing of the program that includes tape save/retrieve capability. His address is given at the start of the text portion of the article in Issue 21.

*Jon Heath* has a tip especially for PC-1 users: If you run out of memory when writing a lengthy number-crunching program, leave off the usual PRINT statements and their memory-consuming captions. Use the ability of the PC-1 to output data when you key in a variable name, such as X, Y, Z or A(12) followed by the ENTER key. It is almost as convenient, since you usually know what each variable represents. And, you can often save enough memory to squeeze in an extra formula or two.

*Thomas Cox* also wants to remind PC-1211/PC-1 users that you can merge programs using the CLOAD1 "NAME" command. However, these merged programs can only be accessed using unique labels, such as A, B, etc., whether being executed (in the DEF mode) or referenced by other routines (such as GOSUB "A"). I seem to recollect that their were some early units that did not have this capability. I guess the best way for you to find out if yours does is to give it a try!

If the new PC-1250/1251 turns out to be popular, there should be lots of software ready to go. Virtually all PC-1211/PC-1 software should run without modification on the new units. What is more, PC-1/1211 programs saved on your own cassette system should load through the CE-125 right into a 1250/1251. (Note that Sharp cautions that tapes prepared on other machines may not load. Thus, the spec's must be pretty tight or they are guarding their flanks!)

Some people seem to be less than happy with the information that comes with the CE-158 RS-232C interface. Seems the instructions on preparing connectors for various types of equipment are less than clear. On the other hand, those that have managed to get gear connected say the unit works great. Maybe a few of those that have gotten good results can share their findings for the benefit of those still struggling?

If you need to do some experimenting with the signal wires going to/from the CE-158 via DB-25 connectors, you might be interested in the Model 51 MINI-PATCH Box from **Remark Datacom, Inc.**, 4 Sycamore Drive, Woodbury, NY 11797, phone (516) 367-3806. They make a "poormans" breakout box priced at \$37.95 that allows you to reconfigure RS-232 connectors using jumper plugs.

*Steven Tripp* reports that over in Japan the PC market is really getting hot with a number of Japanese firms such as Toshiba, Epson and NEC busy supplying the field. One of the most interesting units available over there is NEC's PC-2001 which touts a 2-line, 40-character liquid-crystal display! The unit has a 36K ROM operating system and 8K of RAM expandable to 16K. Some of these units may eventually be exported to the U.S. Steven also indicates that Epson has a dual floppy disk drive available for the HX-20 over in Japan.

**Hewlett-Packard** has dropped prices on several peripherals that can be used with the HP-75C. The HP-82161A Digital Cassette Drive has been reduced from \$550 to \$450, the HP-82162A Thermal Printer/Plotter declined from \$495 to \$450 and the HP-82163A Video Interface (for U.S. equipment) from \$295 to \$225. HP has also announced the introduction of the HP-75 Text Formatter software module to be made available in March at less than \$100.00. The software is intended to make it easy to produce short letters and memos using the HP-75C. In addition to providing formatting options, such as left-and-right justifying, letters may be customized by inserting names and addresses anywhere in the letter. Contact you local HP sales office for additional information regarding this software module.

No new word yet regarding the release of ROM modules for the Sharp PC-1500. About seven modules are past-due and lots of *PCN* readers are anxiously awaiting their arrival.

**Radio Shack** has reduced the price of the PC-2 and the Printer/Plotter Interface. However, Sharp's list price for the PC-1500 is still lower. It looks like the usual price reduction curve is well under way.

— *Nat Wadsworth, Editor*



P.O. Box 232, Seymour, CT 06483